# Inducción: Interfaces.

## Segundo Taller

Elaborado por: Laura Carvajal Segura, Mauricio Loria y Jose David Ortega.

# Contenido

## Motivación

***"Tú puedes, tú eres capaz, tú lo vas a lograr, tú importas, tú haces la diferencia."***
***-Anónimo.***

## Introducción

Normalmente, el segundo proyecto programado busca fomentar la investigación por parte del estudiante sobre cómo realizar interfaz gráfica en Python y cómo consumir diferentes servicios de la web. Este documento está creado para brindarle al estudiante, una perspectiva de cómo realizar interfaces en Python.

**Conceptos Generales**

- **Librería:** Son una gran colección de módulos repletos de código listos para que hagas uso de ellos. Algunas ya vienen incluidas al momento de descargar su IDLE de Python, algunas otras hay que descargarlas por métodos externos (pip de python).
- **Interfaz Gráfica:** Gestiona la interacción con el usuario basándose en relaciones visuales como ventanas, iconos, menús o un puntero.
- **Ventana:** Es un elemento gráfico el cual permite gestionar la interacción con el usuario.
- **Botón:** Permite ejecutar funciones por medio de presionar click.
- **Input:** Permite que el usuario ingrese datos al sistema.
- **Output:** Permite mostrar al usuario datos.
- **POO:** Programación Orientada a Objetos.
    - **Clases:** Definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas.
    - **Objetos:** Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos) los mismos que consecuentemente reaccionan a eventos. Se corresponde con los objetos reales del mundo que nos rodea, o a objetos internos del sistema (del programa).
    - **Atributos:** Tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.
    - **Métodos:** Algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

## Librerías por utilizar:

Se realizará la explicación con dos de las librerías más usadas actualmente para realizar interfaces gráficas en Python.

Estas son:

- Tkinter.
- PyQt.
- EasyGUI.

# Tkinter

## Importando el módulo:

```
from tkinter import *
```

## Creando ventana principal:

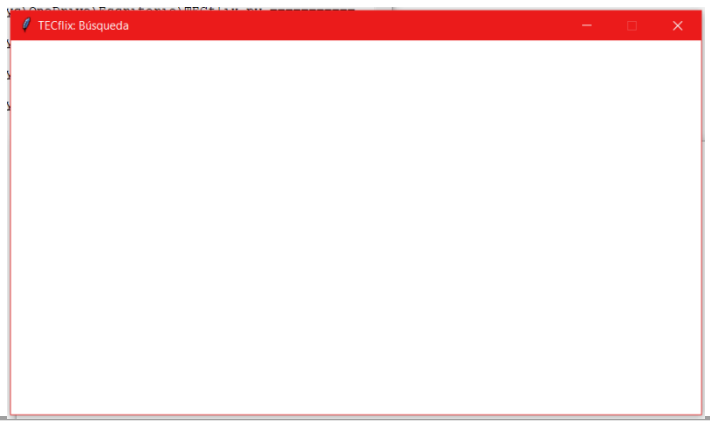
```
#VENTANAS

#Ventanas
principal = Tk()                            # Crea una ventana principal
principal.title('TECflix')                  # Se define el título de la ventana
principal.config(bg="white")                # Le da color al fondo
principal.geometry("710x385+350+200")       # Cambia el tamaño de la ventana
principal.resizable(0,0)                    # Evita que se le pueda cambiar de tamaño a la ventana
principal.mainloop()                        # Este comando permite reconocer a vent1 como ventana principal
```

Resultado:


## Creando ventanas hijas:

```
busqueda = Toplevel(principal)              # Crea una ventana hija, se especifica la ventana a la que pertenece
busqueda.title('TECflix: Búsqueda')         # Se define el título de la ventana
busqueda.config(bg="white")                 # Le da color al fondo
busqueda.geometry("710x385+350+200")        # Cambia el tamaño de la ventana
busqueda.resizable(0,0)                     # Evita que se le pueda cambiar de tamaño a la ventana
busqueda.withdraw()                         # Oculta la ventana para que no se muestre al iniciar la aplicación
```

Resultado:



## Creando botones:

```
# Botón que permite mostrar y ocultar ventanas
                          # ventanaPadre                                          # Función a ejecutar
botonMostrarOcultar = Button (principal,text="Mostrar ventana búsqueda", command=lambda: mostrarOcultarVentana(busqueda, principal))
botonMostrarOcultar.place(x=242,y=2) # El botón es cargado en un lugar establecido por place.
```

Al crear un botón, a este se le asigna una función a ejecutar, en el ejemplo anterior se le asigna la siguiente función.

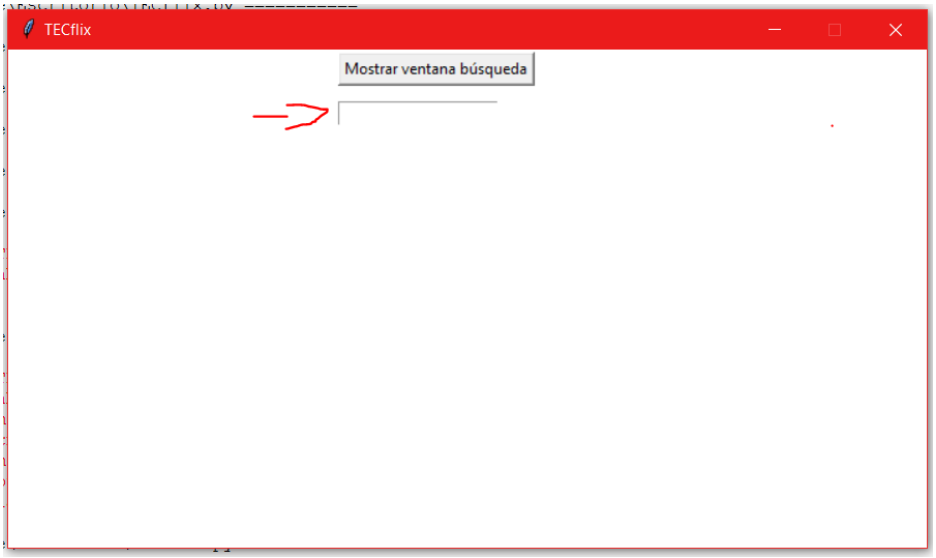
```
    def mostrarOcultarVentana(ventanaMostrar, ventanaOcultar):     # Esta función recibe un objeto de clase ventana
        ventanaMostrar.deiconify()                                 # Se toma el objeto ventanaMostrar y se le aplica el método de mostrar.
        ventanaOcultar.withdraw()                                  # Se toma el objeto ventanaOcultar y se le aplica el método de ocultar.
```

Resultado:


## Creando campos imput:

```
# Input
nombre = StringVar()   # Variable que almacenará el valor recogido

                # ventanaPadre          # Variable
inputNombre = Entry (principal,textvariable=nombre)
inputNombre.place(x=255,y=40)                       # El campo de texto es cargado en un lugar establecido por place.
```

Resultado:



```
# Label
            # ventanaPadre     # Texto a mostrar
label = Label (principal,text="Ingrese su nombre: ")
label.place(x=140,y=40)                  # El label es cargado en un lugar establecido por place.
```

## Creando label:

Los label también funcionan para mostrar información recolectada por medio de input, en este caso en el valor de "text" se sustituye por la variable a mostrar.

Resultado:


## Creando Alerta:

```
# Input
nombre = StringVar()   # Variable que almacenará el valor recogido

                # ventanaPadre          # Variable
inputNombre = Entry (principal,textvariable=nombre)
inputNombre.place(x=255,y=40)           # El campo de texto es cargado en un lugar establecido por place.

botonSaludar = Button (principal,text="Saludar", command=lambda: saludar())
botonSaludar.place(x=255,y=60)      # El botón es cargado en un lugar establecido por place.
```

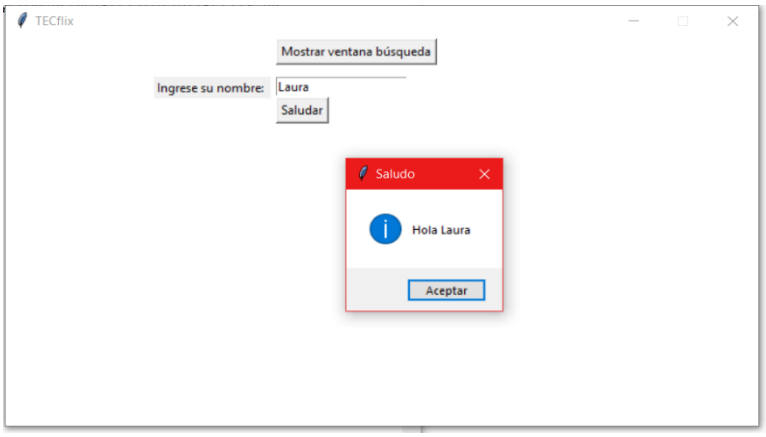
La función saludar es la siguiente:

```
def saludar():
    messagebox.showinfo("Saludo","Hola "+nombre.get())
```

El comando "nombre.get()" obtiene los datos de la caja de texto, ya que a esta caja de texto (inputNombre) se le asignó el "textvariable" de nombre.
El comando message.box.*comando*(**TituloVentana**, **MensajeMostrar**), genera mensajes de alerta.
Existen diferentes alertas de mensajes:

- showinfo(**TituloVentana**, **MensajeMostrar**)
- showwarning(**TituloVentana**, **MensajeMostrar**)
- showerror(**TituloVentana**, **MensajeMostrar**)

Resultado:




## Abriendo archivos:

Primero se creará un botón que invoque a la función que nos permitirá abrir un archivo.

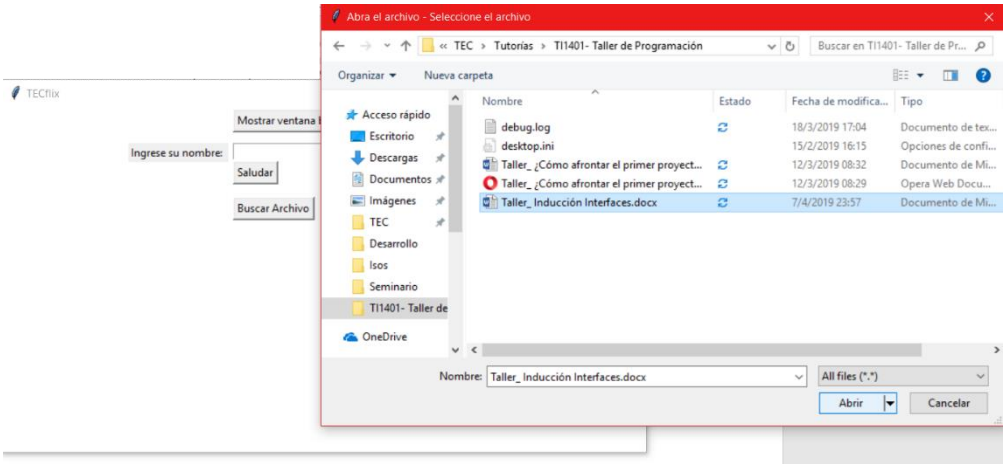
```
botonArchivo = Button (principal,text="Buscar Archivo", command=lambda: buscarArchivo())
botonArchivo.place(x=255,y=100)           # El botón es cargado en un lugar establecido por place.
```

Seguidamente debemos programar dicha función:

```
#Permite que nos salga un pop-up para seleccionar el archivo que queremos abrir
def buscarArchivo():
      extension = ["*.mp4"]
      archivo = eg.fileopenbox(msg="Seleccione el archivo",
                        title="Abra el archivo", default='', filetypes=extension)
      print(archivo)
```

Se indica la extensión del archivo que deseamos abrir. En la variable “archivo” se almacenará la ruta de acceso del archivo seleccionado.

Resultado:




Print de ruta:

```
========== RESTART: C:\Users\danyc\OneDrive\Escritorio\TECflix.py ==========
C:\Users\danyc\OneDrive\Documentos\TEC\Tutorías\TI1401- Taller de Programación\Taller_ Inducción Interfaces.docx
```

## PyQt

Instalación de la librería:

1. Descargar el archivo .whl, correspondiente a la versión de Python instalada en su computadora. Los archivos que contienen “cp37” corresponden a la versión de 3.7 de Python y así consecutivamente con los demás archivos. Los distintos archivos se encuentran en el siguiente link:
   https://www.lfd.uci.edu/~gohlke/pythonlibs/#pyqt4

**PyQt4**, a set of bindings for the Qt4 application framework.
Requires write access to the *site-packages\PyQt4* folder to create *qt.conf*.

PyQt4-4.11.4-cp27-cp27m-win32.whl
PyQt4-4.11.4-cp27-cp27m-win_amd64.whl
PyQt4-4.11.4-cp35-cp35m-win32.whl
PyQt4-4.11.4-cp35-cp35m-win_amd64.whl
PyQt4-4.11.4-cp36-cp36m-win32.whl
PyQt4-4.11.4-cp36-cp36m-win_amd64.whl
PyQt4-4.11.4-cp37-cp37m-win32.whl
PyQt4-4.11.4-cp37-cp37m-win_amd64.whl

2. Abrir la terminal (CMD), navegar a través del comando cd a la carpeta donde se encuentra descargado el archivo .whl, en este caso descargas. Luego a través del comando pip (este debe encontrarse instalado en su computadora) ingresar el siguiente comando: pip install PyQt4-4.11.4-cp37-cp37m-win32.whl.

```
C:\Users\joseo\Downloads>pip install PyQt4-4.11.4-cp37-cp37m-win32.whl
Processing c:\users\joseo\downloads\pyqt4-4.11.4-cp37-cp37m-win32.whl
Installing collected packages: PyQt4
Successfully installed PyQt4-4.11.4
```

**Creación de ventanas:**

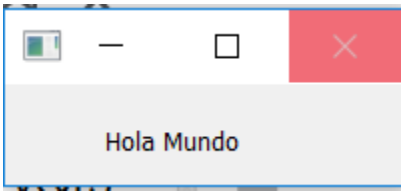
Recordar siempre implementar la siguiente línea de código:

from PyQt4 import QtGui.

Una ventana es un área visual, que muestra la salida de datos y permite los mismos por parte del usuario para uno de varios procesos que se ejecutan simultáneamente.
En el siguiente código se ejemplifica la creación de una ventana que muestra: “Hola Mundo”.

```
import sys
from PyQt4 import QtGui

def window():
    app = QtGui.QApplication(sys.argv)
    w = QtGui.QWidget()
    b = QtGui.QLabel(w)
    b.setText("Hola Mundo")
    w.setGeometry(100,100,200,50)
    b.move(50,20)
    w.setWindowTitle("PyQt")
    w.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    window()
```

Resultado de la ejecución del código anterior:



**Creación de botones:**

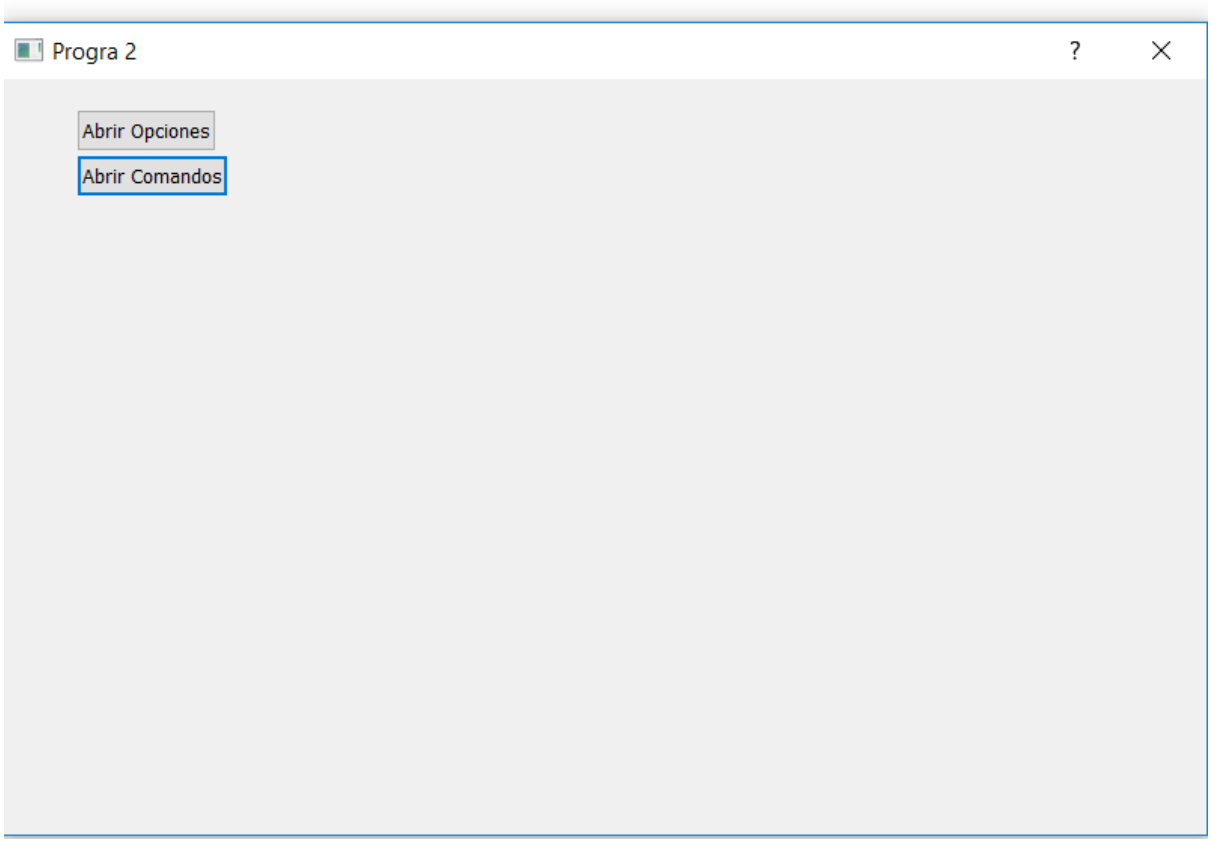
En el siguiente código se ejemplifica la creación de dos botones en una ventana.

```
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *

def window():
   app = QApplication(sys.argv)
   win = QDialog()
   b1 = QPushButton(win)
   b1.setText("Abrir Opciones")
   b1.move(50,20)
   b1.clicked.connect(b1_clicked)

   b2 = QPushButton(win)
   b2.setText("Abrir Comandos")
   b2.move(50,50)
   QObject.connect(b2,SIGNAL("clicked()"),b2_clicked)

   win.setGeometry(400,400,800,500)
   win.setWindowTitle("Progra 2")
   win.show()
   sys.exit(app.exec_())

def b1_clicked():
   print ("Estas son las opciones: ")

def b2_clicked():
   print ("Caja de Comandos del Sistema")

if __name__ == '__main__':
   window()
```

A diferencia del código anterior, se incluyen dos funciones nuevas, la primera de ellas es: “b1_clicked” esta función tiene como objetivo imprimir en pantalla: “Estas son las opciones” al ser presionado. La siguiente función implementada: “b2_clicked” tiene una funcionalidad similar, solo que al ser presionado imprimirá en pantalla: “Caja de Comandos del Sistema”.

El siguiente es el resultado de ejecutar el código anterior:

```
Estas son las opciones:
Caja de Comandos del Sistema
```



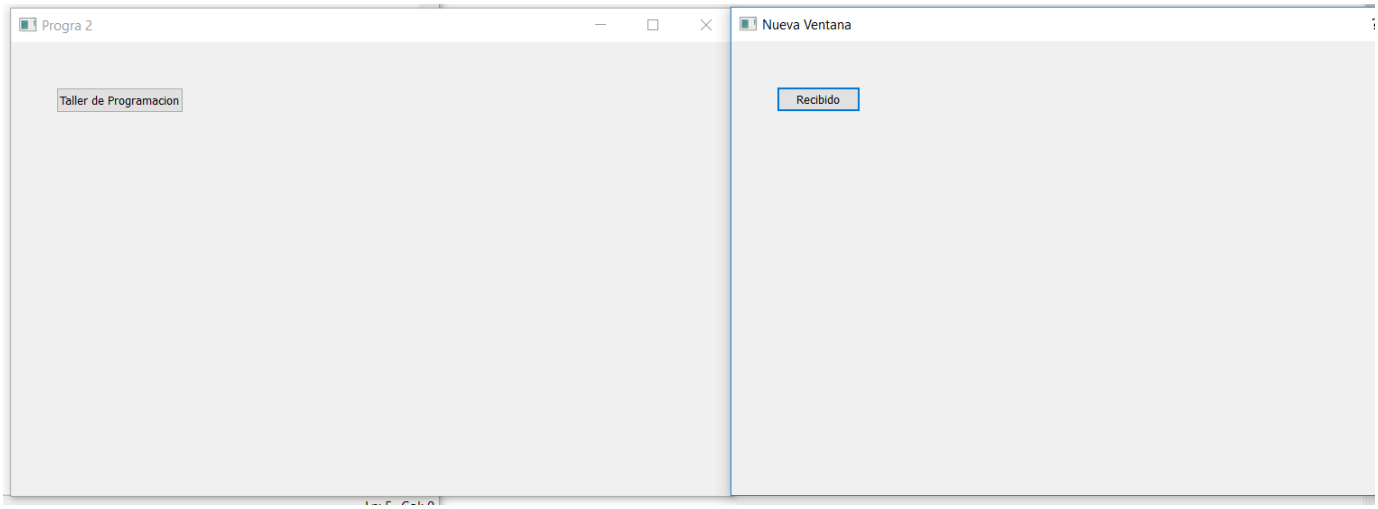

Como se observa se imprime en consola un mensaje, según el botón que se presione.

**Abrir nuevas ventanas:**

El siguiente código ejemplifica como abrir nuevas ventanas, en este caso al hacer click sobre el botón “Taller de Programación” se abrirá una nueva ventana titulada “Nueva Ventana”.

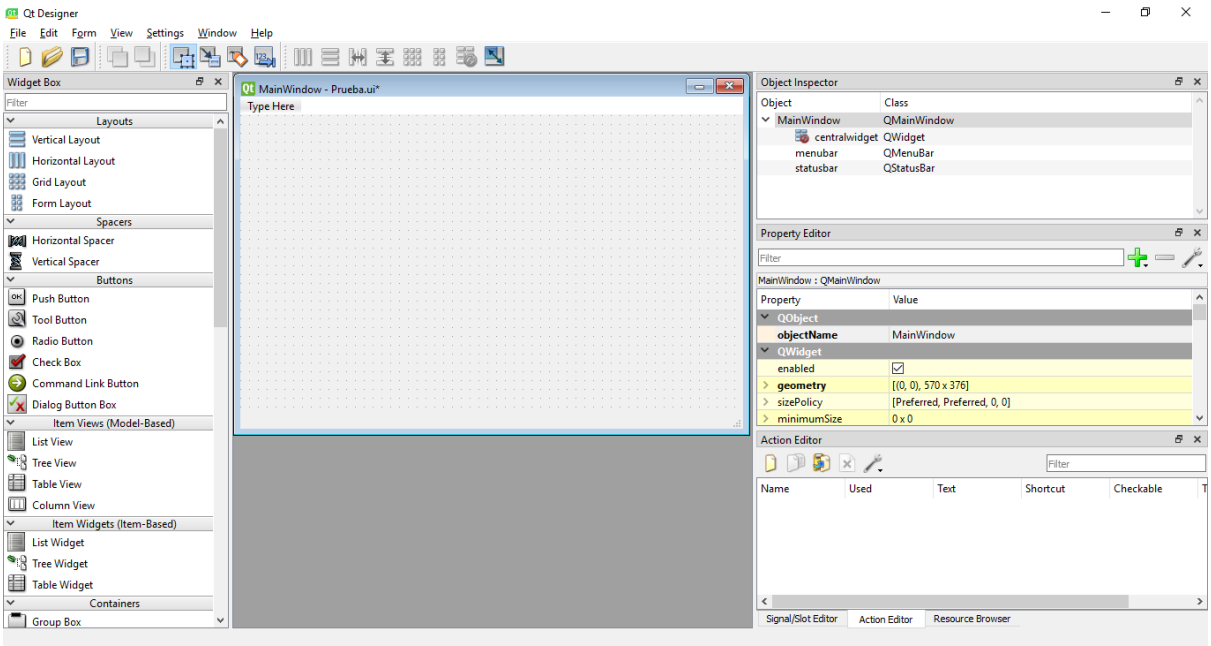
```
import sys
from PyQt4.QtGui import *
from PyQt4.QtCore import *

def window():
    app = QApplication(sys.argv)
    w = QWidget()
    b = QPushButton(w)
    b.setText("Taller de Programacion")
    b.move(50,50)
    w.setGeometry(400,400,800,500)
    b.clicked.connect(showdialog)
    w.setWindowTitle("Progra 2")
    w.show()
    sys.exit(app.exec_())

def showdialog():
    d = QDialog()
    b1 = QPushButton("Recibido",d)
    d.setGeometry(400,400,800,500)
    b1.move(50,50)
    d.setWindowTitle("Nueva Ventana")
    d.setWindowModality(Qt.ApplicationModal)
    d.exec_()

if __name__ == '__main__':
    window()
```

El método “connect” recibe como parámetro la función que crea la nueva ventana en este caso: “showdialog”, este método a su vez es parte del método “clicked” del label “b”.
La ejecución del código anterior tiene como resultado:



## PyQt Designer

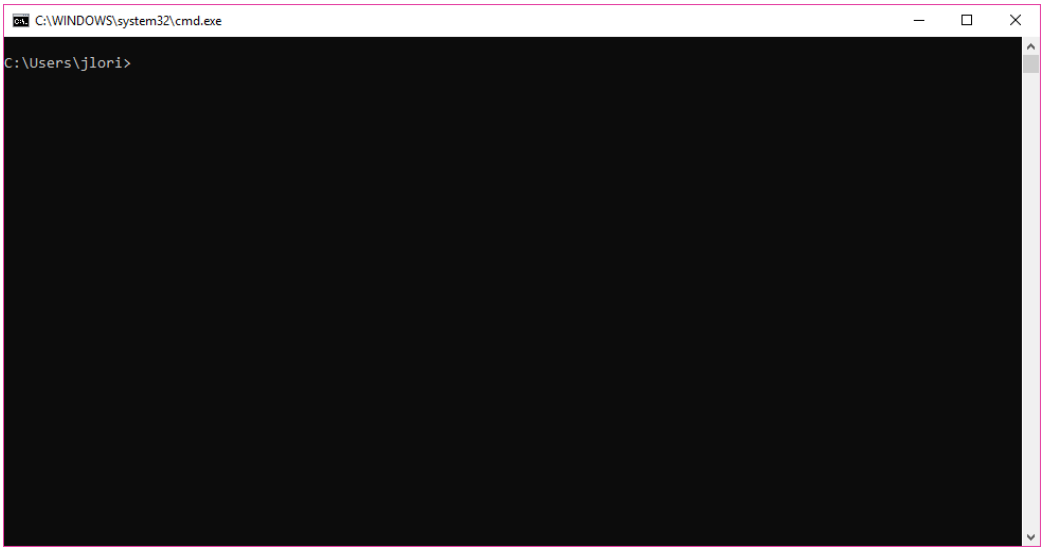
Esta interfaz permite el diseño de ventanas, agilizando el proceso de creación de estas.



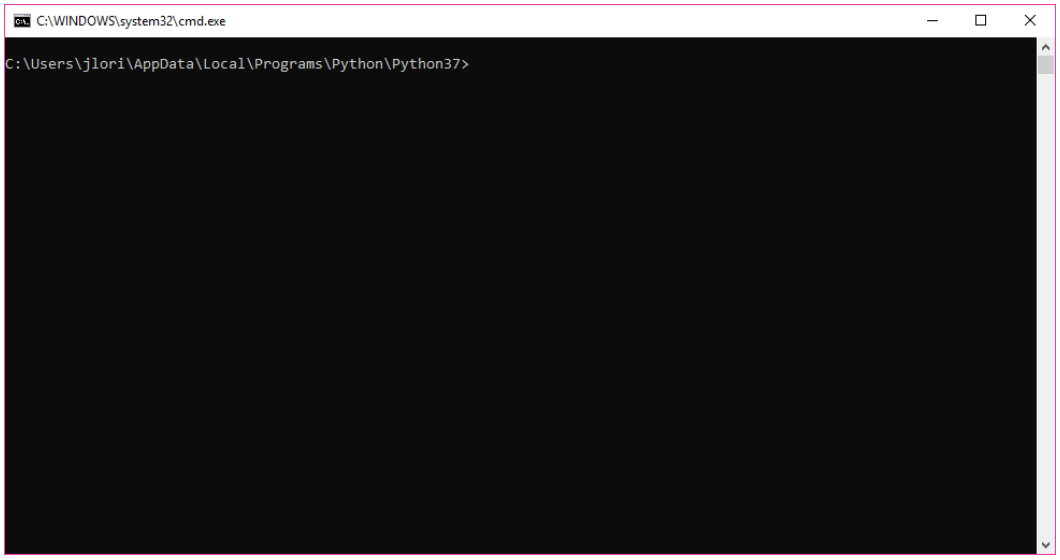

### Generar el código de la interfaz:

Posterior al diseño de la interfaz gráfica, es necesario generar el código para poder incorporar la funcionalidad respectiva a los diferentes botones, listas, etiquetas, cuadros de texto, entre otros. Por el cual la librería incluye un archivo llamado pyuic5.py para convertir los archivos de la interfaz los cuáles tienen la terminación .ui a archivos .py. Eso lo podemos hacer de la siguiente forma:

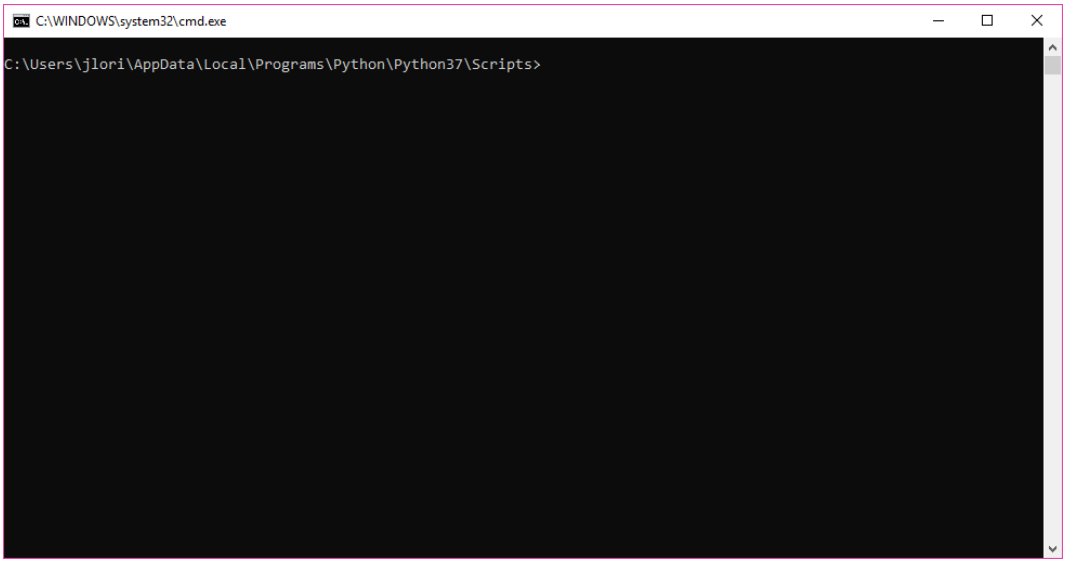
1. **Desde la terminal**:



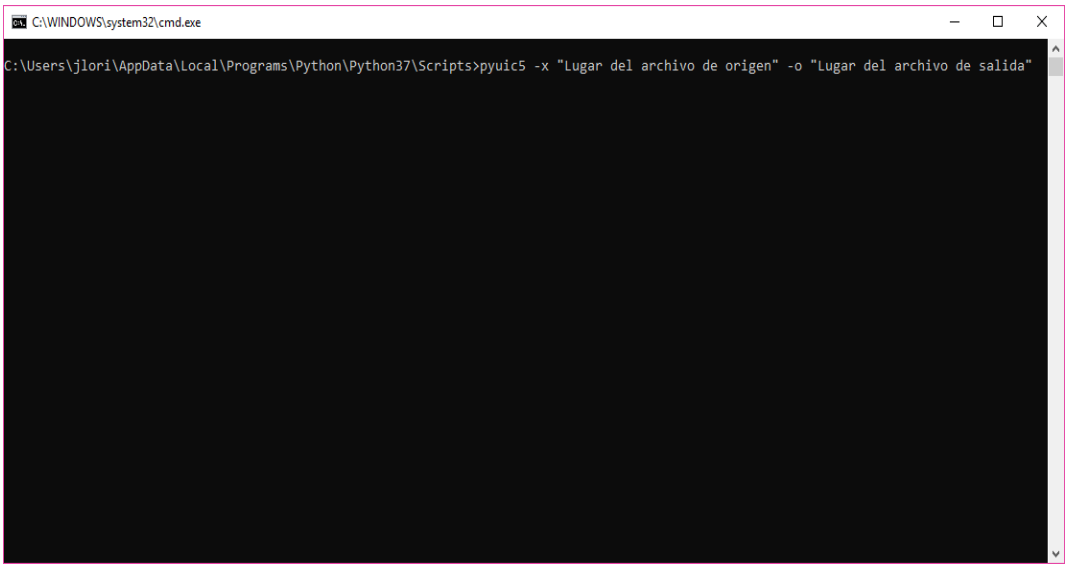
2. Desplazarse a la carpeta en donde se encuentra instalado Python en la computadora:




3. Desplazarse a la carpeta en donde se encuentran instalados las diferentes librerías:




4. Utilizar el siguiente comando con las siguientes banderas:



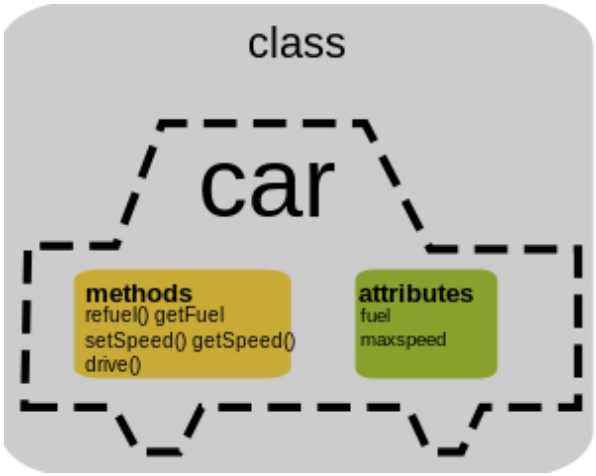

Nota**: Podría ser el mismo comando “pyuic” pero con la terminación en la versión de PyQt que hayan descargado.

5. Así se vería un ejemplo del código generado por el comando anterior:

```python
# -*- coding: utf-8 -*-

#
# Created by: PyQt5 UI code generator 5.11.3
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(516, 332)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.printHello = QtWidgets.QPushButton(self.centralwidget)
        self.printHello.setGeometry(QtCore.QRect(210, 130, 75, 23))
        self.printHello.setObjectName("printHello")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 516, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

        self.printHello.clicked.connect(self.imprimirHolaMundo)

```

## POO: Programación Orientada a Objetos

Es un paradigma de programación el cual viene a innovar la forma de obtener resultados. Una clase se puede definir de las propiedades y comportamiento de un tipo de objeto concreto. Podemos pensar de un una clase como una entidad, así por ejemplo de un carro el cual tiene sus propios atributos como lo es el motor, modelo, color, así como también ciertas funciones las cuales puede realizar como avanzar, retroceder, entre otros. Podemos llamar esas funciones como el comportamiento de la clase.




Mediante el uso de estas clases es posible crear objetos, los cuales pueden tener sus características únicas. Es posible para nosotros identificar entidades las cuales comparten ciertas características y comportamiento, o que suelen ser muy similares. Existen varias técnicas como lo es la abstracción, herencia, encapsulamiento, polimorfismo, acoplamiento, cohesión. Las cuales son usadas para hacer buenos diseños de clases, mantener alejados los olores y mantener los principios del programador.
Python es un lenguaje el cual hace uso de este paradigma, tiene su propia sintaxis para crear clases, a continuación un ejemplo de cómo se vería la clase Car de la imagen anterior:

```
class Car:

    def __init__(self,fuel,maxspeed,color):
        self.fuel = fuel
        self.maxspeed = maxspeed
        self.color = color

    def setSpeed(self,speed):
        self.maxspeed = speed

    def getSpeed(self):
        return self.maxspeed

    def reFuel(self,quantity):
        self.fuel += quantity

    def getFuel(self):
        return self.fuel
```

La función o también conocido como método “__init__” es el constructor de la clase, este nos permite asignar propiedades al objeto en su creación. El “self” en la sintaxis de Python permite hacer referencia a atributos propios de la clase permitiendo así, poder accesarlos desde diferentes métodos dentro de la clase al igual que desde objeto creado.

### Binding Functions:

Para enlazar funciones con los botones podemos crear métodos dentro de la clase que nos permita realizar acciones, una vez un botón ha sido presionado, obtener la información de cuadros de texto, establecer el contenido de etiquetas esto lo podemos lograr de la siguiente manera.

Algunas funciones comunes de los diferentes componentes de PyQt:

1. **Agregar funcionalidad a un botón:**

Se puede agregar el siguiente método a la clase de la ventana:

```
def imprimirTextbox(self):
    print(self.inputText.text())
```

Como pueden ver recibe el parámetro self, para referirse a los atributos propios de esa clase. Este método permitirá imprimir el texto ingresado en la consola al presionar un botón.

```
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(516, 332)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.printHello = QtWidgets.QPushButton(self.centralwidget)
        self.printHello.setGeometry(QtCore.QRect(210, 130, 75, 23))
        self.printHello.setObjectName("printHello")
        self.inputText = QtWidgets.QLineEdit(self.centralwidget)
        self.inputText.setGeometry(QtCore.QRect(190, 170, 113, 20))
        self.inputText.setObjectName("inputText")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 516, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

        self.printHello.clicked.connect(self.imprimirTextbox)
```

Aquí se puede apreciar como el botón printHello, llama a la función clicked y connect para asociarla al método creado previamente de la clase imprimirTextbox, es importante agregar el uso de self antes de llamar al método para referirse al método de hace clase.

1. **Obtener el texto ingresado de un cuadro de texto:**

```
def imprimirTextbox(self):
    print(self.inputText.text())
```

2. **Establecer el valor de un cuadro de texto o etiqueta:**

```
def setTextbox(self,texto):
    self.inputText.setText(texto)
```

3. **Agregar elementos a un listbox (listwidget):**

```
def agregarListWidget(self,elemento):
    #Usualmente se agregan Strings
    self.listWidget.addItem(elemento)
```

4. **Eliminar elementos de un listbox (listwidget):**
   Este método elimina todos los elementos del listwidget

```
def eliminarElementosListWidget(self):
    self.listWidget.clear()
```

   Nota**: La función .takeItem(index) elimina y retorna un elemento en específico del listwidget

5. **Obtener el valor de un radiobutton o checkbox:**

```
def obtenerRadioButton(self):
    presionado = self.radioButton.isChecked()
    #presionado es una variable que almacena un boolean
    return presionado
```

**Ejercicio:**

Realice un programa llamado “Calculadora” con interfaz gráfica que permita introducir dos valores desde la interfaz gráfica y los sume o reste (según indique el usuario) es decir debe crear 2 botones: Uno llamado sumar, que muestre el resultado de la suma de los valores introducidos por el usuario y un botón restar, que muestre el resultado de la resta de los valores introducidos por el usuario. El resultado de las operaciones se puede mostrar tanto en la consola o si lo desea en la interfaz gráfica.